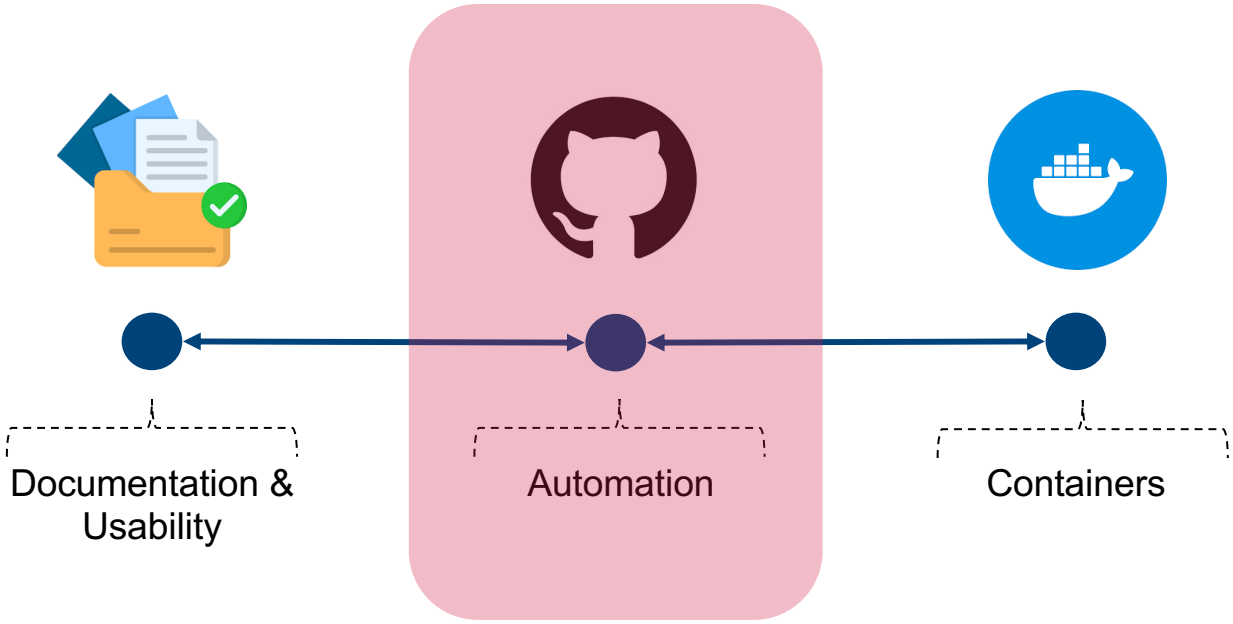
Best practices for reproducibility 🎉



Version control

- As an aid for reproducibility
- Provides a framework for collaborating, particularly on code
- Allows you to restore previous versions and compare between versions
  - Ever wish you could "Undo changes" to say, the version 2 months ago? Now you can!
  - Note: this functionality is much more powerful than “Track Changes” in Word or file recovery options in Google Drive

- code-final.R, code-final-final.R, code-final-final-for-real-this-time.R
- file.R, file.R (1), file.R (2), file.R (3)

Source edited by Data Lab Reproducibility Workshop

git for version control

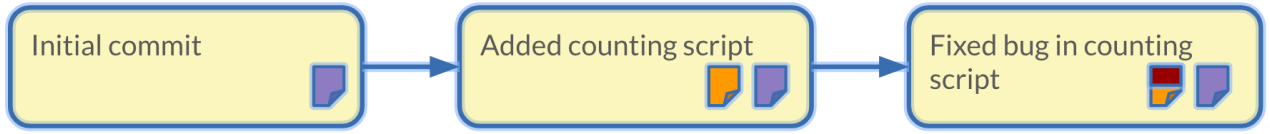
- **git** is an open-source software that performs version control
  - implemented as a set of command line tools
  - GitKraken is a GUI that (hopefully) makes git a bit easier to use
    - Other available GUIs: SourceTree, GitHub Desktop

- GitHub.com is one of many websites that can host your repositories
  - Others include gitlab, bitbucket, gitbucket…
  - Using a host like GitHub means that you:
    - have a backup at all times
    - can work on your code from different computers
    - collaborate with other more easily

Source by Data Lab Reproducibility Workshop

git terminology

- A repository (repo) is a directory that git is managing
  - It looks like a regular folder from your perspective
  - There can be multiple copies of the repository, often on different machines
- Files that git knows about are tracked, i.e., they are under version control
- We take snapshots called commits to record the state and history of tracked files.

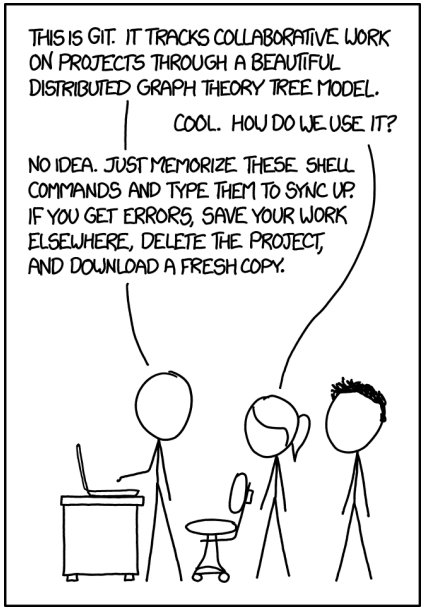


Source by Data Lab Reproducibility Workshop

Pearls of wisdom ⋆ ｡°⋆🐚⋆❀｡° ⋆

- git can be tricky, and the feelings of frustrations you ~~may~~ will have are normal!
- You will make mistakes.

  That's ok! So do I, and so does everyone else!
- Remember: You have to start somewhere.

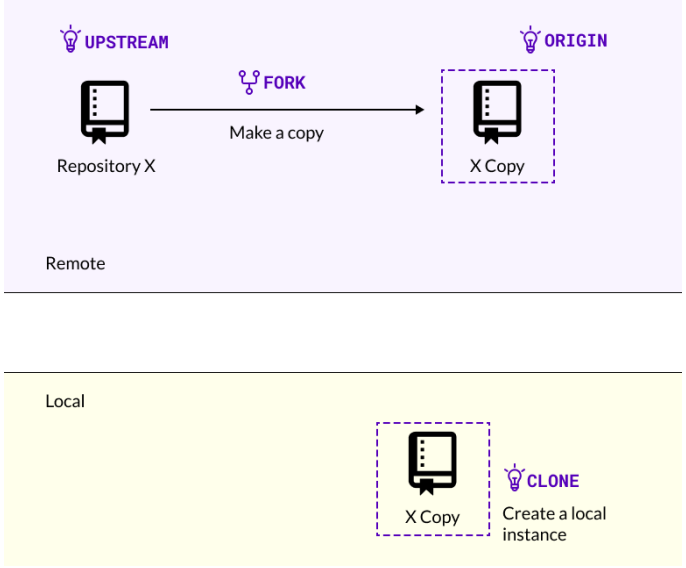


https://xkcd.com/1597/

Source by Data Lab Reproducibility Workshop

Forking a repository on GitHub

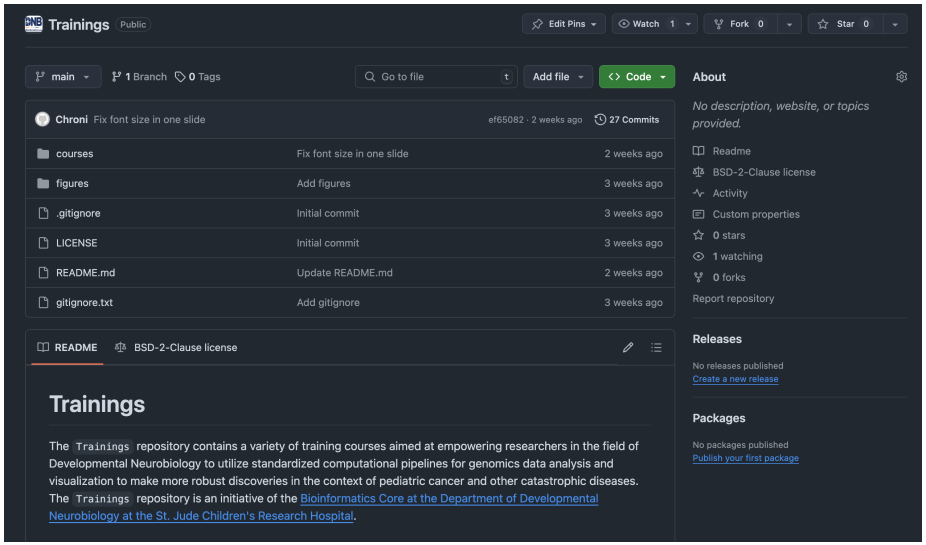
- Forking allows you to make your own copy of somebody else's repository
  - Anything that happens in the forked repository will not affect the "upstream" repository
  - You will be able to make whatever changes you want
  - Connections to what happens upstream will still exist, which can be handy
- Note that with your own project repositories, you would not need this step - you can just make a repository directly



Source by Data Lab Reproducibility Workshop

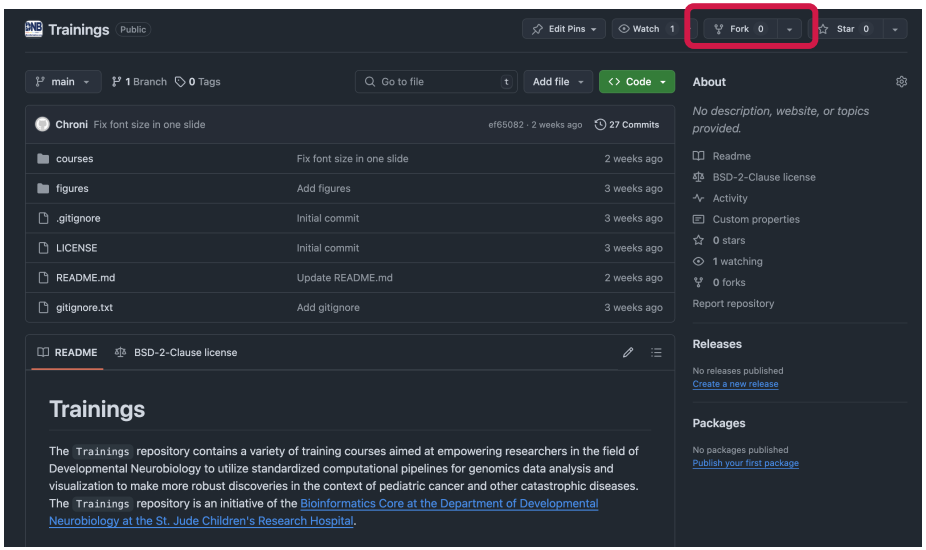
Forking a repository on GitHub

Go to https://github.com/stjudeDNBBinfCore/trainings/
(make sure you are logged into GitHub)



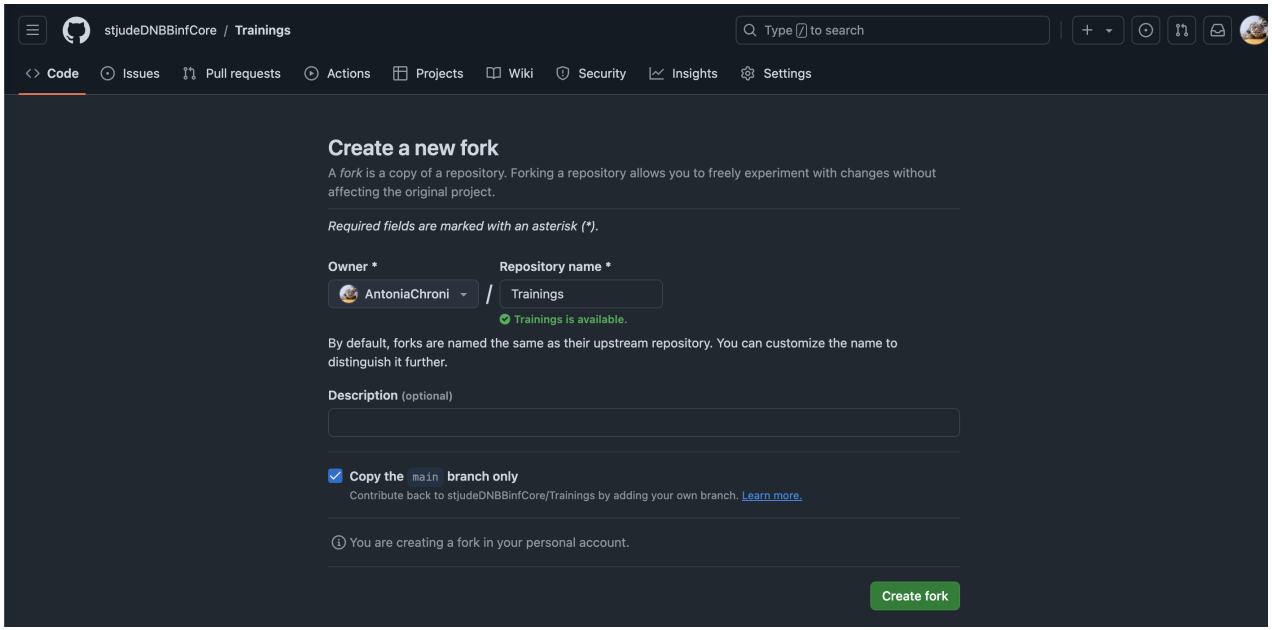
Forking a repository on GitHub

Click the “Fork” button



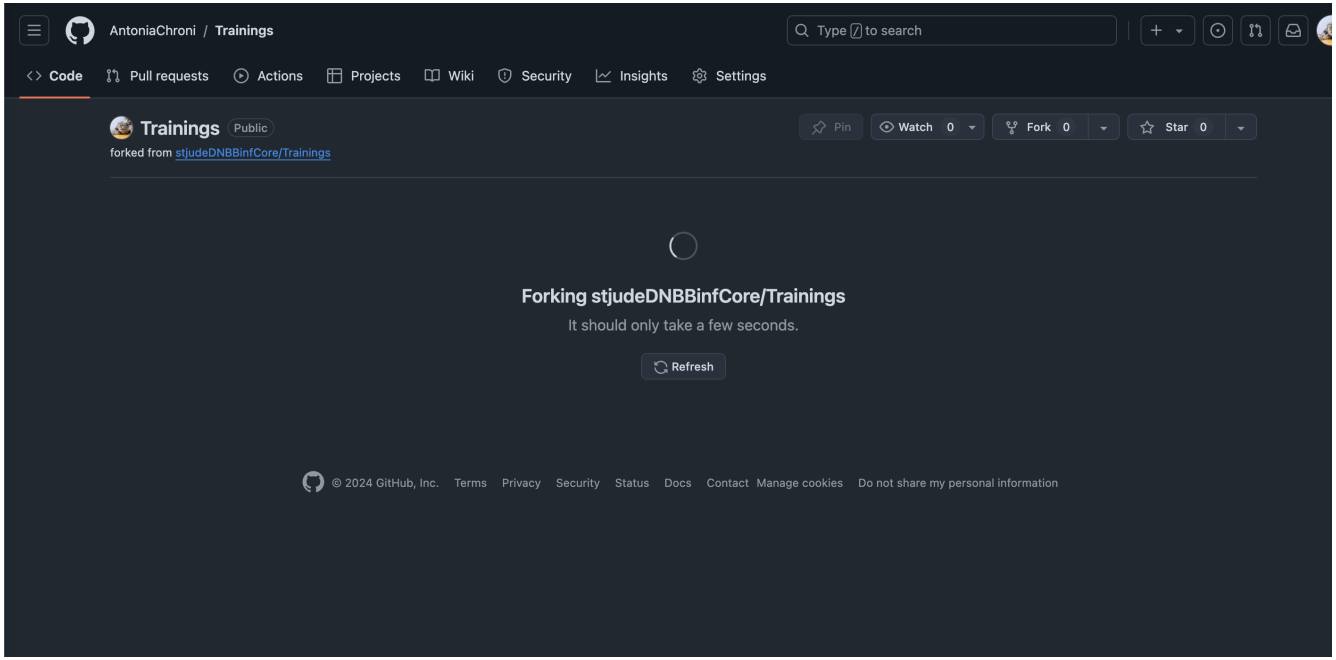
Forking a repository on GitHub

Change the name if you like, but probably not; click “Create fork”



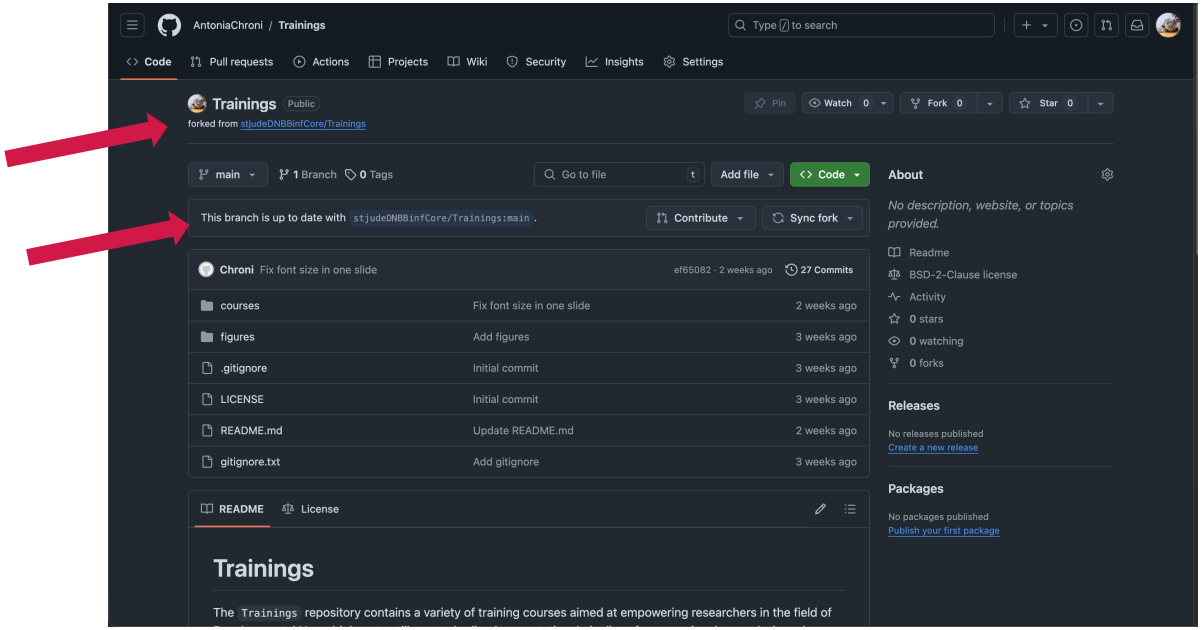
Forking a repository on GitHub

Wait…



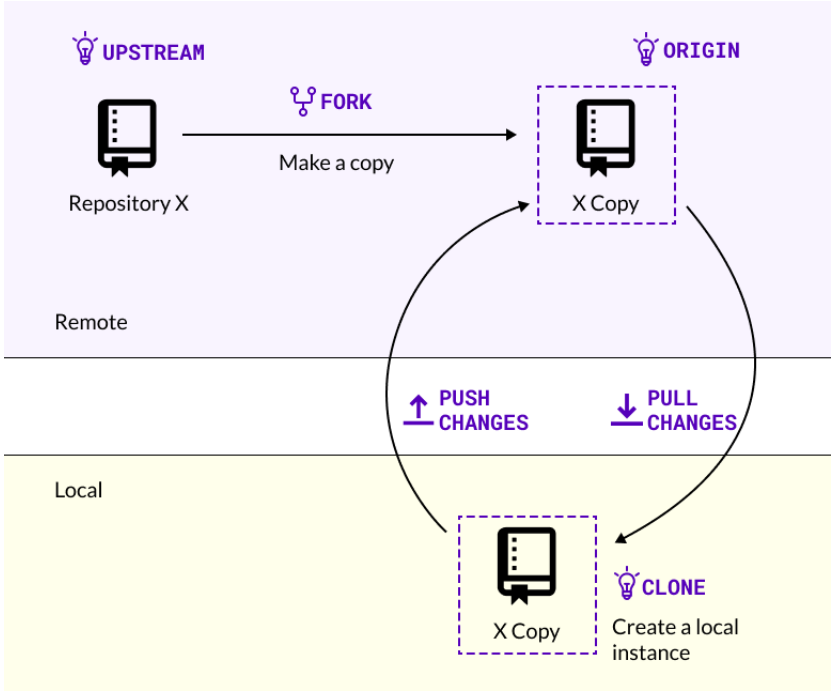
Forking a repository on GitHub

Enjoy your new repo!



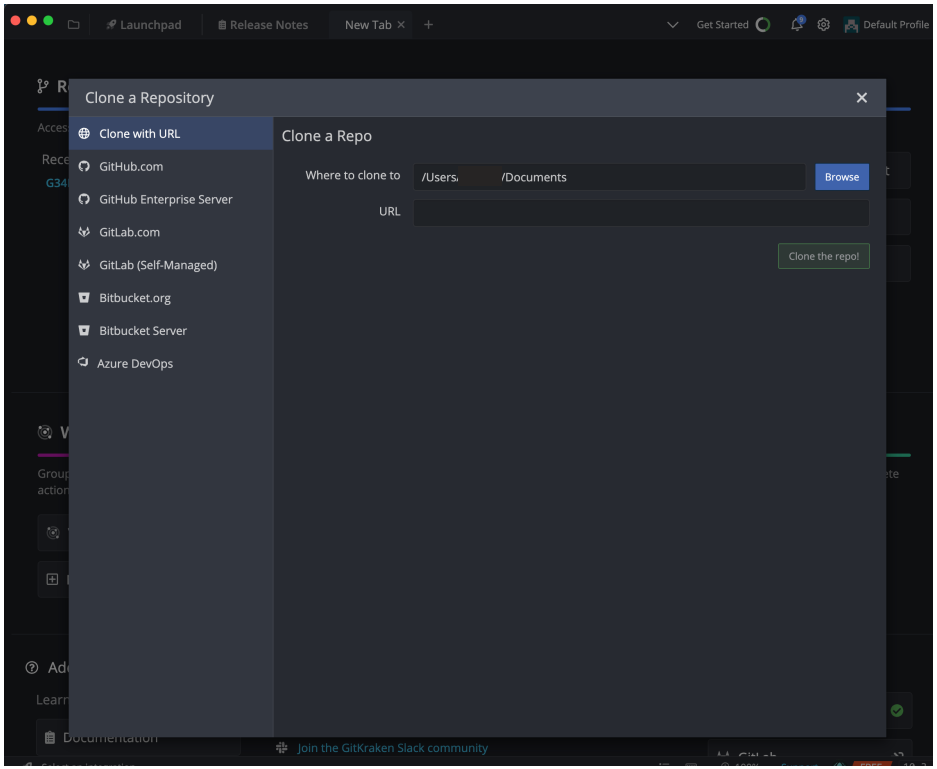
Cloning your repository on your computer

- Having a repository on GitHub is great, but you really want a copy on your own computer!
  - This is true whether it is a repo you made "from scratch" or one you forked
  - You can also go the other direction, starting a repo on your computer, then adding it to GitHub.
- Cloning can happen at the command line or through a GUI like GitKraken



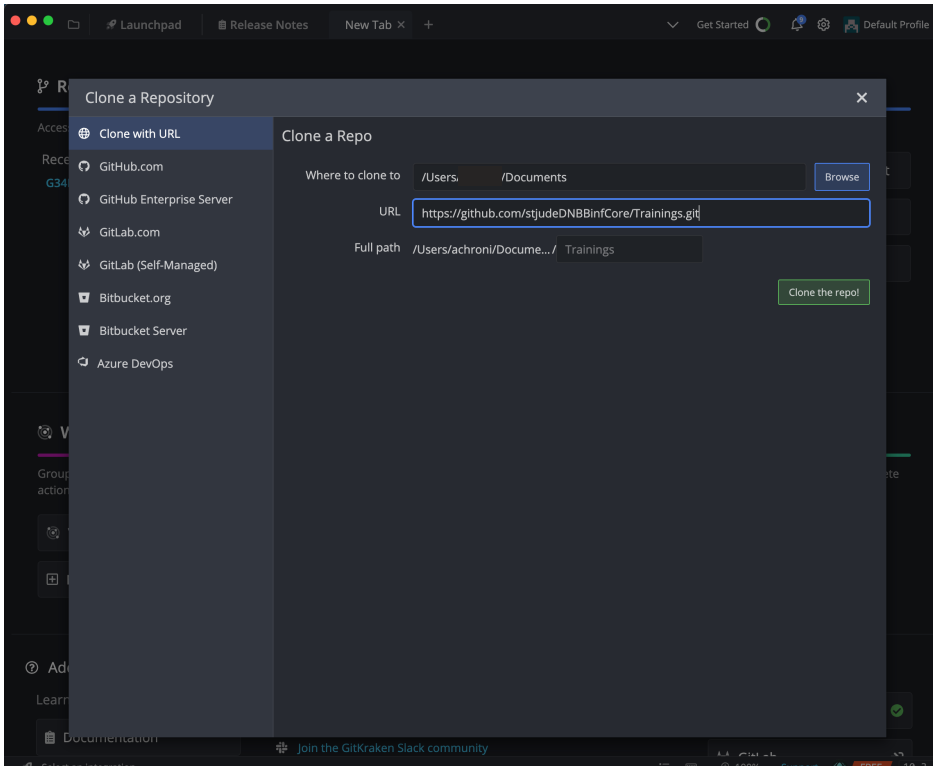
Cloning in GitKraken

- Choose the destination
  - “Where to clone to”
- “Clone with URL” copied from GitHub

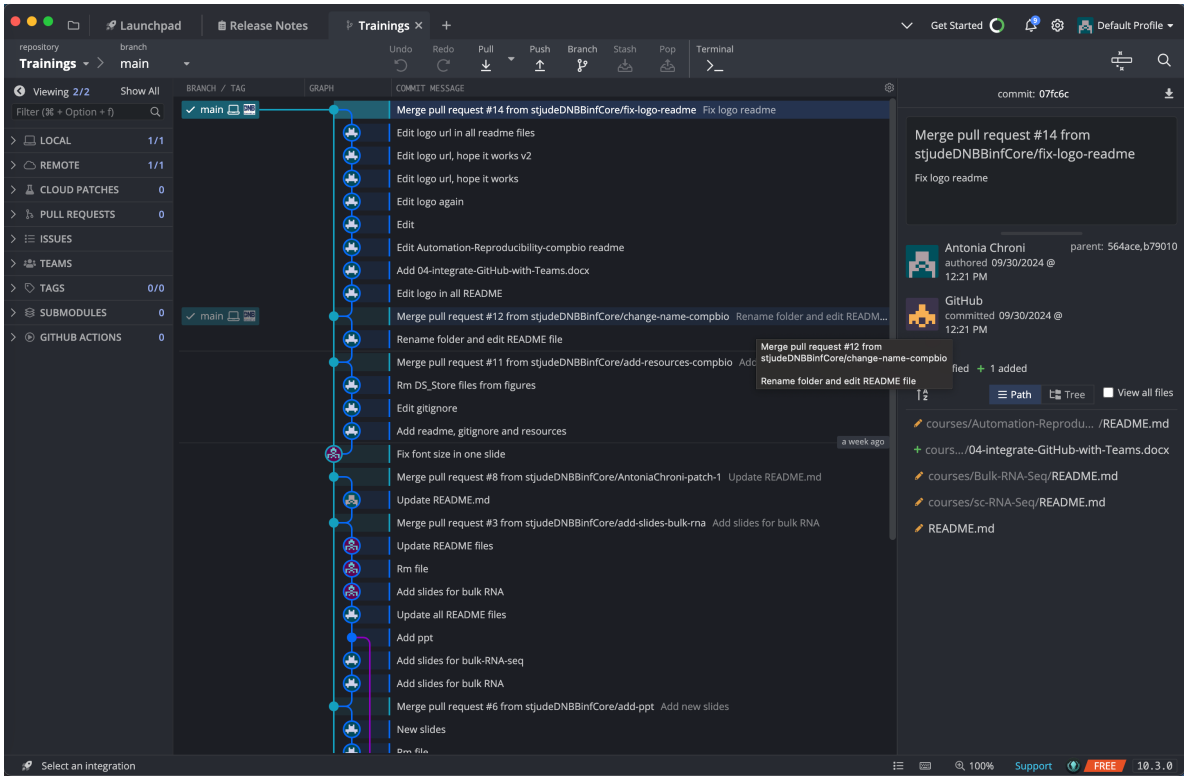


Cloning in GitKraken

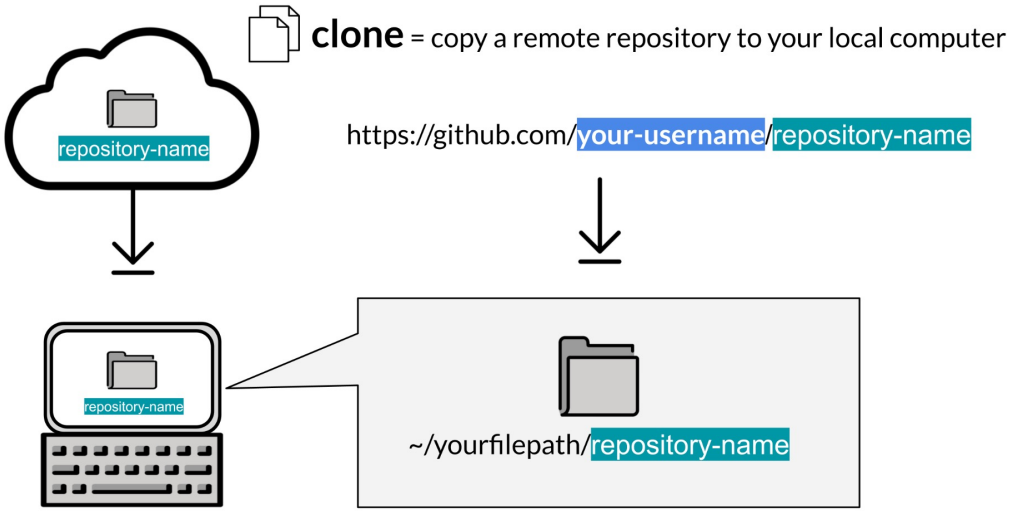
- Choose the destination
  - “Where to clone to”
- “Clone with URL” copied from GitHub
- OR select the GitHub.com repository from within GitKraken!



Cloning in GitKraken



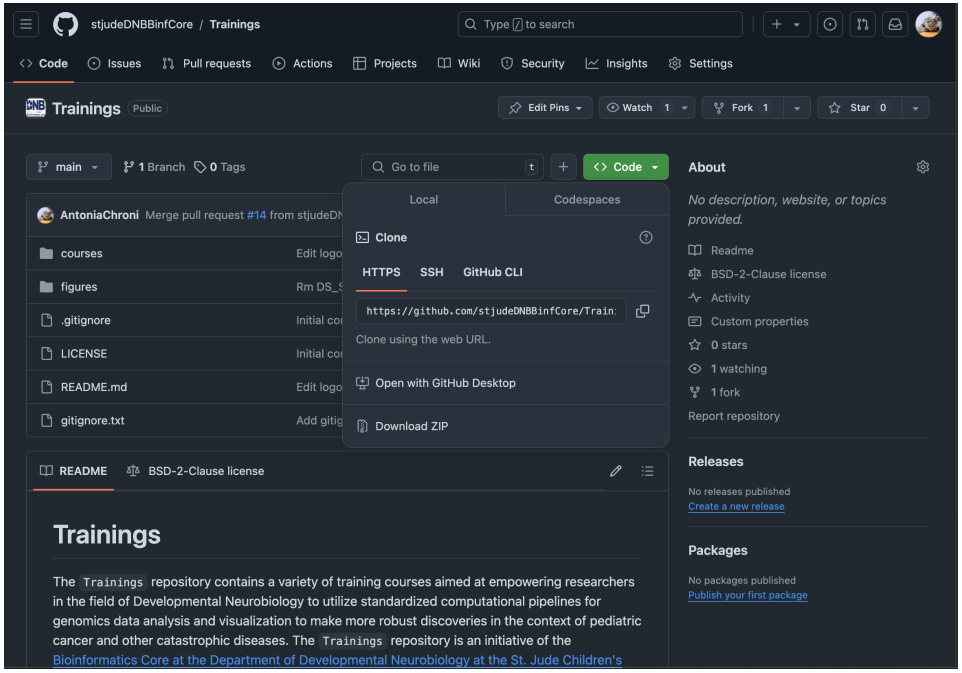
git clone



Source by Advanced Reproducibility in Cancer Informatics

Cloning at the command line

- Find the repository URL and copy it:
  - Click the “Code” button
  - Copy the URL under “Clone”
  - the URL will end with a **.git** extension



Cloning at the command line

- Navigate to the location where you want to put the repository with `cd`
- git clone <URL-from-GitHub>

```
Demo % git clone https://github.com/stjudeDNBBinfCore/Trainings.git
Cloning into 'Trainings'...
remote: Enumerating objects: 195, done.
remote: Counting objects: 100% (108/108), done.
remote: Compressing objects: 100% (87/87), done.
remote: Total 195 (delta 40), reused 63 (delta 21), pack-reused 87 (from 1)
Receiving objects: 100% (195/195), 77.09 MiB | 33.27 MiB/s, done.
Resolving deltas: 100% (81/81), done.
Demo %
```

Cloning at the command line

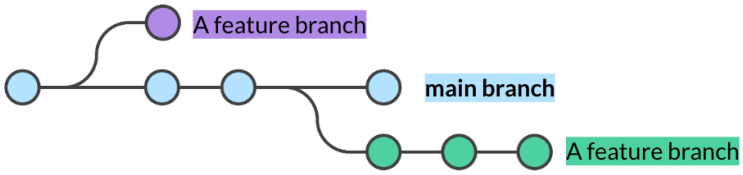
- Use cd to move into your new repository and ls to look around
  - cd Trainings
  - ls -la (include all of the hidden files)

```
Trainings % ls -la
total 32
drwxr-xr-x@  9          staff   288 Sep 30 19:30 .
drwx------@ 12          staff   384 Sep 30 19:33 ..
drwxr-xr-x@ 13          staff   416 Sep 30 20:00 .git
-rw-r--r--@  1          staff   371 Sep 30 19:30 .gitignore
-rw-r--r--@  1          staff  1308 Sep 30 19:30 LICENSE
-rw-r--r--@  1          staff  1960 Sep 30 19:30 README.md
drwxr-xr-x@  5          staff   160 Sep 30 19:30 courses
drwxr-xr-x@  5          staff   160 Sep 30 19:30 figures
-rw-r--r--@  1          staff   819 Sep 30 19:30 gitignore.txt
```

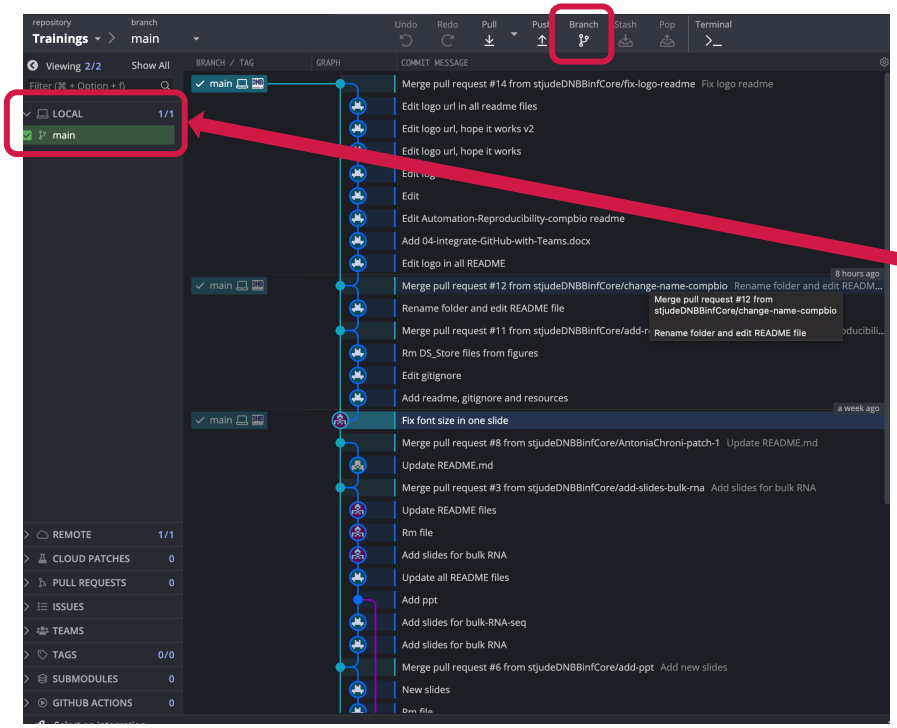
What is a branch?

- Branches are like "repositories within repositories" 🤯
- Useful when you want to make changes (maybe experimental!) but you don't want to break the rest of your code
  - You can always switch back to a "clean" branch!
- Keep related changes together
  - All commits for a given new analysis or "feature" can be made within the same branch for easier tracking
  - Helps you to identify which commits are relevant to a given analysis
- If you wreck code in a branch, you've *only wrecked that branch!* Just delete it!
- Branches provide a great framework for collaboration and team science

branch = a unique working copy of file changes of a GitHub repository. A branch can be local and remote.



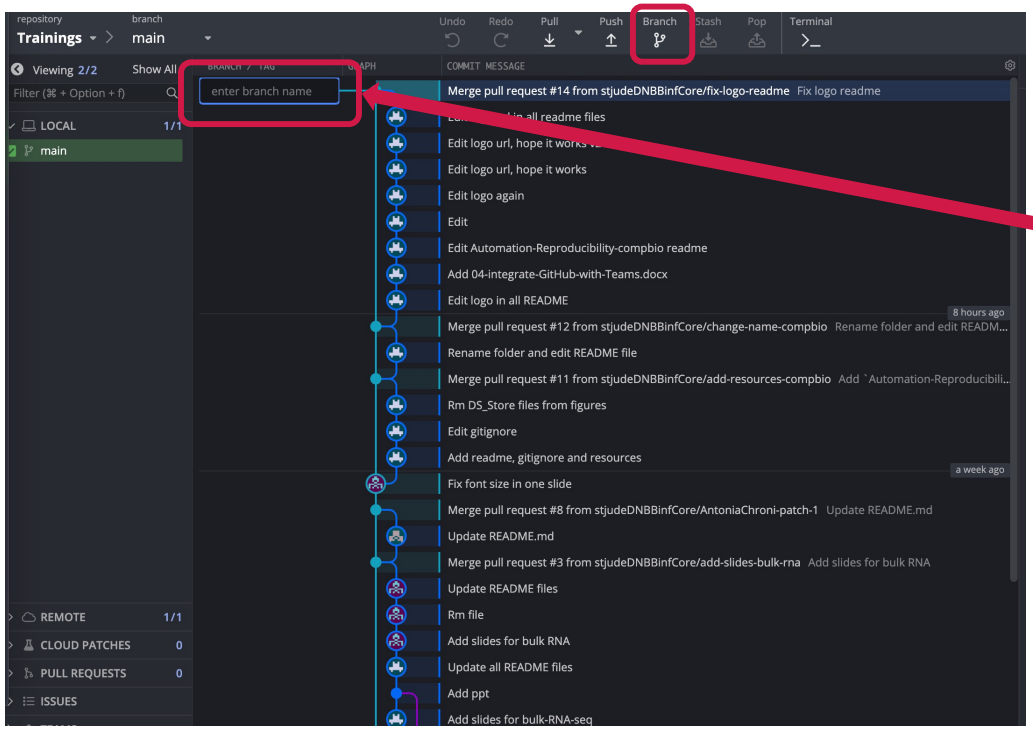
Create a new branch



⚠️ Every time you create a branch, this will be created off of the branch that you are currently in!!! ⚠️

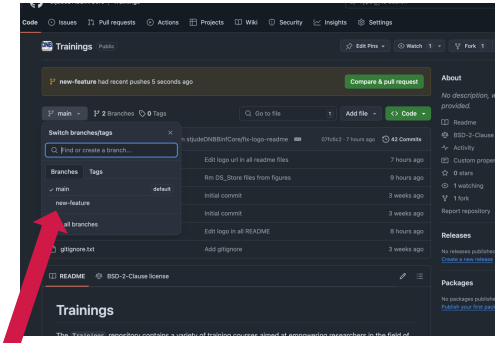
- If you are at the main, then it will be based on it.
- If you are at another feature branch, then it will be from that → nested branches that will have to be ordered and merged in that order as well

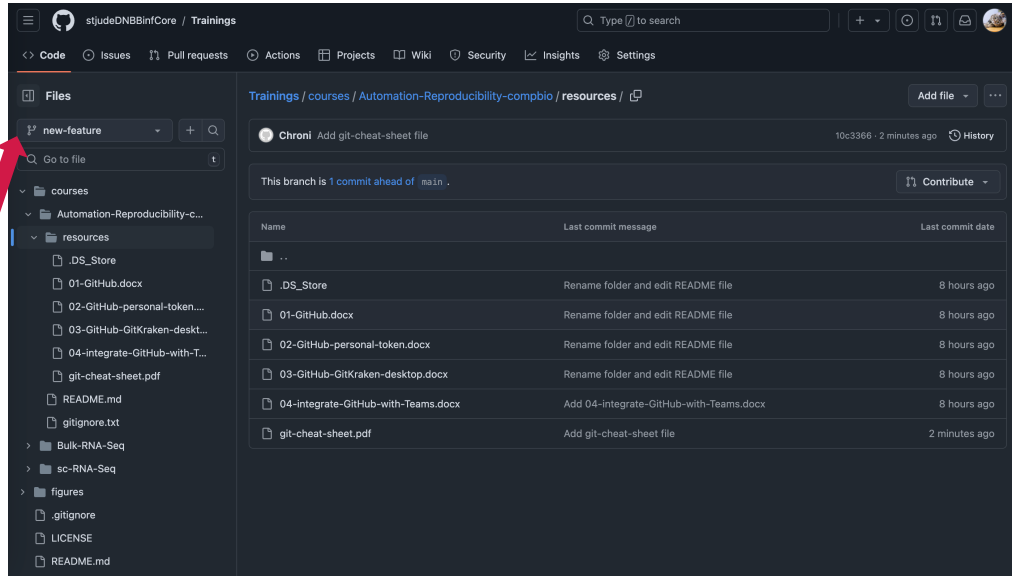
Create a branch



1. Add your branch name/click enter
2. This will automatically create your feature branch. This is the same as: git branch <branch_name>
3. GitKraken will automatically checkout your new –branch: git checkout <branch_name>
4. This is your working branch now and you can add files etc

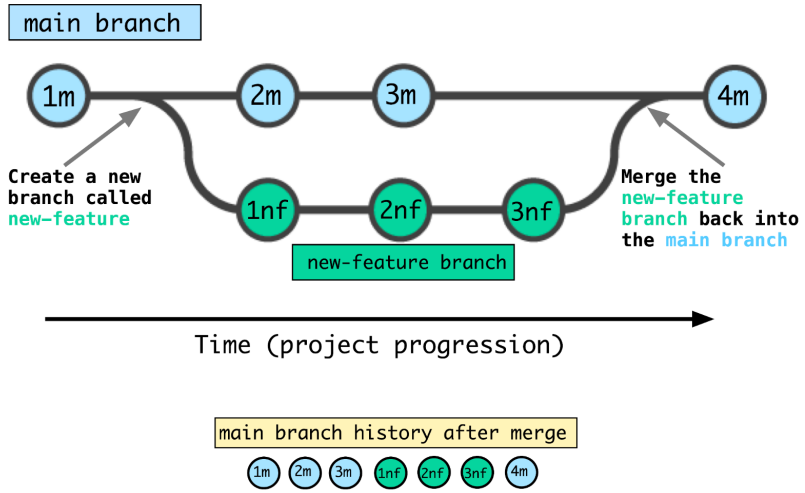
Viewing branches on GitHub





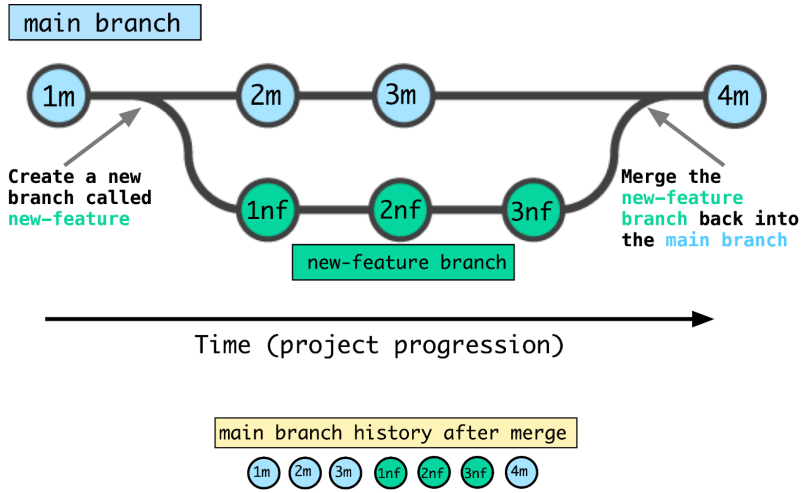
Many (nested) branches and steps!

1. Create new branch to add/delete code/address an issue
2. Add/Delete/Stage the changes you want to be part of the next commit: Best practice to stage one changed file at a time to avoid problems (you may also hear this step referred to as "add")
3. Commit your code changes with an informative message
4. Pull/push code from local to remote repo (and vice versa)
5. Create PR/issue to start code review process
6. Once everything is complete, then merge branch to main 🎉



Before merging though…

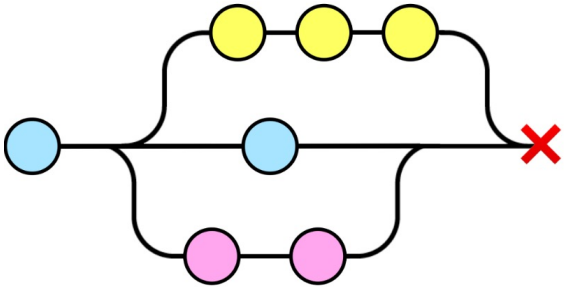
- When you are working *independently*, merging directly into main on your own is probably fine!
- When you are working *collaboratively*, it can get dangerous because your collaborators won't be in the loop, and conflicts will probably emerge.



Merge conflicts can happen!

- If a given file has been heavily modified, git may not be able to merge the different file versions across branches together automatically.
  - This is especially a problem if there are many branches floating around getting merged into each other
  - Imagine if each of these branches modified the same file in drastically incompatible ways……..
- Do not panic, everything will be okay!

⚠ You will have to manually fix the merge conflicts! ⚠



VS Code also has helpful git integration!

```
1   Biospecimen_ID  primary_site
    Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
2   <<<<<<< HEAD (Current Change)
3   BS_HZV4WDTB Frontal Lobe|Parietal Lobe
4   =======
5   BS_HZV4WDTB Frontal and Parietal Lobes
6   >>>>>>> update-metadata-fields (Incoming Change)
7   BS_E1SWA20C Peripheral Whole Blood
8   BS_KB9GJDCS Cerebellum/Posterior Fossa;Ventricles
9   BS_2EJWS3SD Peripheral Whole Blood
10  BS_6YMJ621P Skull
```

How much stage is too much for a commit?

- Smaller units of work are generally better *for you!*
- Imagine you are looking in your code history. Which commit message is more helpful?
  - All the work I did on Tuesday 😱
  - Modified notebook to change plot size 🙌
- Which scenario is easier to look through?
  - 1-3 changed files in a commit
  - 25 changed files in a commit

`git` commands cheat sheet

- git status
- git pull
- git push
- git branch
- git branch <new_branch>
- git branch -d <localBranchName> # delete branch locally
- git push origin --delete <remoteBranchName> # delete branch remotely
- git add <file> or git add .
- git rm <file>
- git commit –m “<what_you_did>”
- `git checkout` command is very complicated. It's so complicated that someone finally, as of Git 2.23, split it into two separate commands 🙌
  - git switch, which does the "select other branch or commit to have checked out" operation; and
  - git restore, which does the "update some files in index and/or working tree" operation

`git` commands cheat sheet

- git log <path/filename> # find last git commit for a specific file
- checkout to your branch- myBranch
  - git checkout myBranch
- get latest code from master branch to your branch
  - git pull origin main OR
  - git rebase main
- git checkout HEAD . # if you want to go back one commit, explore more undo commit here

---

- More commands here, and
- `./resources/git-cheat-sheet.pdf`

Do not worry, I'll show you how to do all this soon 😀

Not all files are meant to live under version control

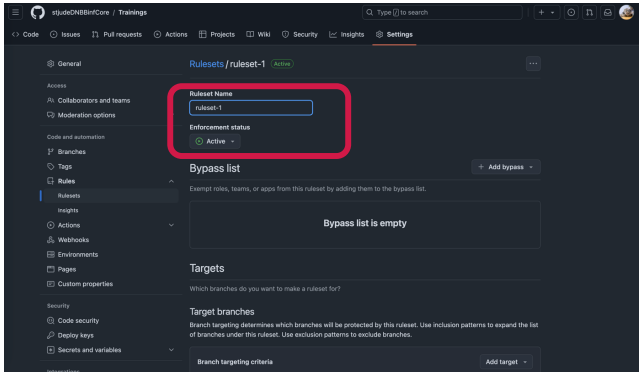
- Sometimes we have files inside the repository directory that we very much do not want to be part of version control.
  - Large files; GitHub cannot generally handle files >= 100 MB. Git Large File Storage lets you store large files on a remote server such as GitHub (see also here for a potentially good solution).
  - Unreleased/private data files, including PII/PHI data - even in a private repository, don't do it!
  - Other sneaky culprits like .DS_Store files on MacOS

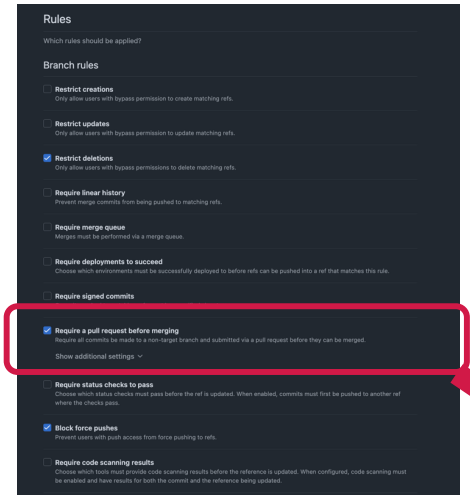
- It's pretty easy to mess up and accidentally stage/commit these files, and Git loves to complain about unstaged files.

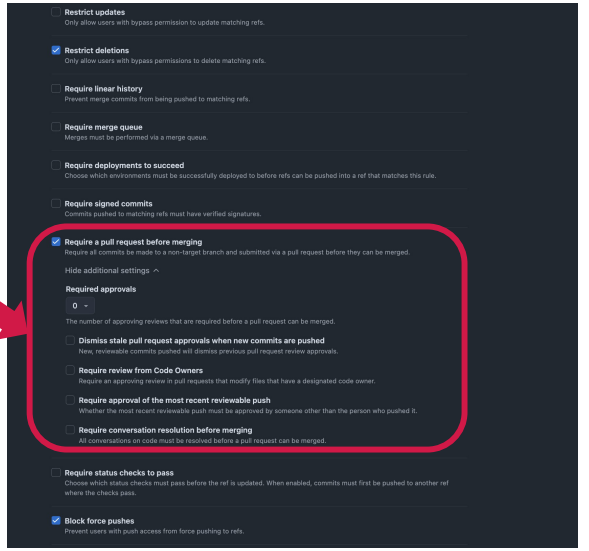
Using `.gitignore` files

- `.gitignore` files are *hidden files* that tell git to ignore certain files
  - `.gitignore` will make Git stop telling you about untracked changes
  - `.gitignore` will prevent you from staging/committing these files by accident
    - 🚨Word to the wise:🚨 Include files ASAP in your .gitignore before you accidentally commit them or you'll have trouble get rid of them…
- You can have many `.gitignore` files in a repo (in different directories) and/or a single .gitignore at the top of your repo that all subdirectories "inherit"
  - You can also have a *global* file in `~/.gitignore` that will apply to all repos on your computer (Requires a little more setup, though)!
  - But caution: Your collaborators probably don't have that file.

Using `ruleset`







A reminder: Everyone agrees Git is tricky.

- You will make mistakes
  - That's ok! So do I, and so does everyone else!
  - Git and GitKraken error messages will try to help you.
- But sometimes you will just want to

Dangit, Git!?!

Git is hard: messing up is easy, and figuring out how to fix your mistakes is impossible. Git documentation has this chicken and egg problem where you can't search for how to get yourself out of a mess, *unless you already know the name of the thing you need to know about* in order to fix your problem.

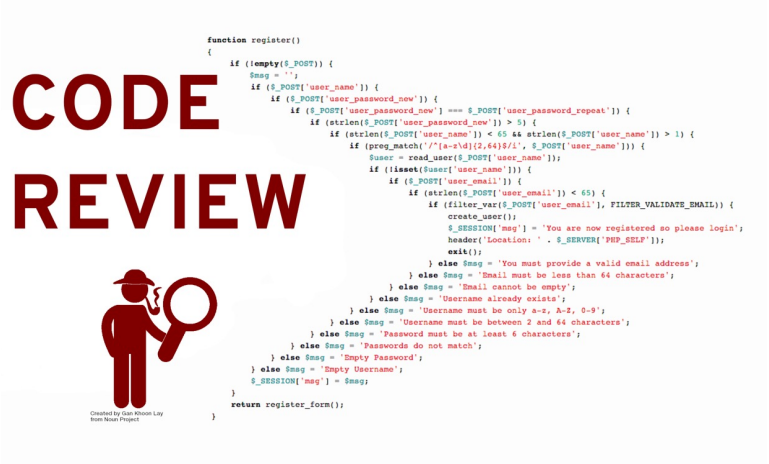
So here are some bad situations I've gotten myself into, and how I eventually got myself out of them *in plain english*.

Dangit, I did something terribly wrong, please tell me git has a magic time machine!?!

https://dangitgit.com/en

Code review

- Code review helps boost the accuracy and reproducibility of the analysis.
- Everyone involved in the process will learn something new.
- Effective code review brings so many benefits not only to your project quality but also your communication skills through fostering a learning atmosphere in the team.
- It begins with the creation of a pull request/issue.
- Successful and efficient code review is born out of quality communication.
- Even if you end up being the only person who will review your own code, writing these things out is still very helpful and highly recommended.



https://openpracticelibrary.com/practice/code-review/

Code review rules for authors

- Be explicit on your PR for what kind of feedback is needed. Let them know that before they waste their time digging into the code line-by-line.
  - Are you still in the early stages and looking for a bigger picture review?
  - Are there specific areas of the code you are having trouble with or are unsure about? Send a link to the specific lines in GitHub you are asking about.
  - Are there results that are surprising, confusing, or smell wrong?
  - Are you in the later stages and looking for detailed nit-picky review?
  - Are you looking for feedback on the results or methods?

Code review rules for authors

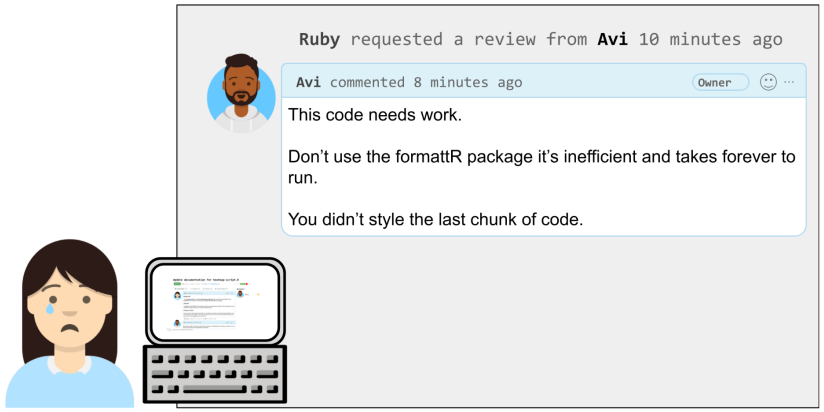
- Try to make sure your pull requests aren't too long! Code reviewing fatigue is very real. If you send a reviewer thousands of lines of code to review it will be very overwhelming to review or understand.
- Alternatively, when you create a new branch try to set a very intentional (and relatively small) goal you would like to achieve with your upcoming pull request. Keeping your pull requests small and focused on one task at a time will not only help your reviewers but also will help yourself feel more accomplished and organized.

Code review rules for reviewers

- Identify areas in the code and documentation that are opportunities for improvement.
- Communicate your questions and concerns effectively and in a way that creates a positive atmosphere.
- Determine solutions collaboratively in a way that allows for a learning as well as a long term improved product.
- What to look for:
  - Does the analysis answer the question it's asking? Are the methods it uses to do so appropriate?
  - Is the code clear and readable? Does it contain a healthy amount of comments and documentation for individuals not familiar with the project to understand generally what is going on?
  - Is the code efficient with computational resources? (Are there areas it's a bit too greedy with memory usage?)
  - Does the code stick to the style and conventions of this project?
  - Are there alternate scenarios where the current strategy might fail? (depending on the likelihood of this use case, this may be an instance for a new issue and for it to be addressed in a different pull request).

Code review rules for reviewers – how to communicate

- Try to be empathetic to the learning process! You are both working on this project together – assume you both want the best out of this project. If something seems wrong, work together to find a solution, don't ever waste time on placing blame.



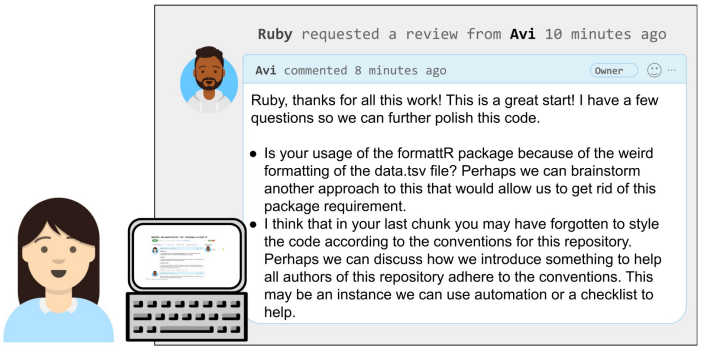
Code review rules for reviewers – how to communicate

- Writing code is hard and writing quality code is even harder. Let's be appreciative/suggestive and highlight things we liked as well! 😀

  - *Nice work Alice, I really learnt a lot from this.*
  - *Great Job! Bob*

> I suggest you use an ArrayHelper getValue method here because of its error handling capability instead of accessing the value directly You could even go further by giving an example: $a = $b['key']; would raise an error if key is not set but $a = ArrayHelper :: getValue(b, 'key'); would return a null value if key is not set.

> Nice Job! Alice. I suggest we create an interface for this service so other substitute services can implement the interface as well, this would enable us change to a different service with very minimal efforts when the need arises. What do you think?



Empathy is an important part of effective code review!

- This interaction reminds us that effective code review is steeped in empathy from both sides. Authors need to appreciate the time and effort the reviewer is spending to help them; while reviewers need to be sensitive to the amount of effort put in by the author already.

More resources 🎉

- GitHub Docs
- Git cheat-sheet
- GitHub Automation for Scientists – explains GitHub Actions Fundamentals
- GitKraken Git GUI Tutorial For Beginners
- Happy Git and GitHub for the user
- awesome-code-review
- Comments during Code Reviews